1. Tempo disponibile 120 minuti.
2. Non è possibile consultare appunti, slide, libri, persone, siti web, ecc.
3. Scrivere in modo leggibile, su ogni foglio, nome, cognome e numero di matricola.
4. Le soluzioni agli esercizi che richiedono di progettare un algoritmo devono:
   - spiegare a parole l'algoritmo (se utile, anche con l'aiuto di esempi o disegni),
   - fornire e commentare lo pseudo-codice (indicando il significato delle variabili),
   - calcolare la complessità (con tutti i passaggi matematici necessari),
   - se l'esercizio ammette più soluzioni, a soluzioni computazionalmente più efficienti e/o concettualmente più semplici sono assegnati punteggi maggiori.

1. Calcolare la complessità $T(n)$ del seguente algoritmo `mystery`:

```
algoritmo mystery(A: Int[1..n]) --> Int
    h = new BinaryHeap()
    for i = 1..n
        h.insert(A[i])
        for j = 1..i
            h.insert(A[i] * j)
        endfor
    endfor
    return h.findMax()
```

Soluzione L'algoritmo `mystery` esegue operazioni di inserimento e ricerca del massimo in un heap, che rispettivamente hanno complessità logaritmica nella dimensione dell'heap, e costante. Per calcolare la complessità dovremo quindi quantificare il numero di esecuzioni di tali operazioni. Viene effettuata una sola operazione di ricerca del massimo, quindi la complessità deriva dal costo degli inserimenti. Il numero di inserimenti risulta essere $n+\sum_{i=1}^{n} i = n+\frac{n\times(n+1)}{2} = O(n^2)$. Si noti che al momento del k-esimo inserimento, l'heap avrà dimensione $k-1$, quindi la complessità di tale inserimento sarà $O(\log k)$. Quindi la complessità computazionale degli $O(n^2)$ inserimenti nell'heap sarà: $O(\sum_{i=1}^{n^2} \log i) = O(\log(\prod_{i=1}^{n^2} i))$ applicando la regola del logaritmo di prodotti. Concludendo, avremo $T(n) = O(\log(n^2!))$.

2. Si consideri una tabella hash di dimensione $m = 10$ inizialmente vuota e usata per memorizzare chiavi intere. Le collisioni sono gestite tramite concatenamento con inserimento in testa e i valori hash delle chiavi sono calcolati con il metodo della divisione.

   Mostrare il contenuto della tabella e il fattore di carico dopo ognuna delle seguenti operazioni, eseguite in ordine: INS(10), INS(20), INS(11), INS(17), INS(26), INS(30), DEL(20), DEL(11), INS(21) INS(27), INS(77), DEL(26).

Soluzione Le immagini alla fine del documento mostrano come cambia la tabella e il fattore di carico dopo le operazioni. Per brevità sono state accorpate alcune operazioni di inserimento o cancellazione eseguite su liste diverse.

3. Si devono preparare i sacchetti della merenda per una classe di k bambini. Ogni bambino mangia un diverso numero di panini, bisogna quindi preparare sacchetti personalizzati. Abbiamo inizialmente $k' \geq k$ sacchetti posizionati in fila, uno dopo l'altro, ognuno contenente un proprio numero di panini. Dobbiamo agire sui sacchetti in fila in modo tale che alla fine rimangano solo k sacchetti, ognuno con il numero giusto di panini per il corrispondente bambino; ovvero, il primo sacchetto deve contenere i panini per il primo bambino, il secondo

sacchetto quelli per il secondo bambino, e così via. Per prepararli si procede nel seguente modo: si considerano i sacchetti secondo il loro ordine, dal primo all'ultimo della fila, e ogni volta che si considera un sacchetto lo si toglie dalla fila, oppure lo si lascia in fila eventualmente inserendo o estraendo panini dal sacchetto. I panini estratti dai sacchetti precedenti, o presenti nei sacchetti precedentemente tolti dalla fila, possono essere inseriti in successivi sacchetti. Non sempre è possibile preparare i sacchetti: ad esempio, se abbiamo tre sacchetti con rispettivamente 3, 2, e 4 panini, non è possibile preparare due sacchetti rispettivamente da 6 e 3 panini (mentre sarebbe possibile prepare due sacchetti rispettivamente da 1 e 6 panini, ad esempio togliendo due panini dal primo sacchetto, togliendo dalla fila il secondo, ed inserendo due panini nell'ultimo sacchetto). Progettare un algoritmo che dati due vettori di numeri naturali `IN` (di lunghezza k') e `OUT` (di lunghezza k), che rappresentano rispettivamente il contenuto dei sacchetti iniziali (ovvero `IN[i]` indica il numero di panini del sacchetto in posizione i nella fila iniziale) ed i panini richiesti dai bambini (ovvero `OUT[j]` indica il numero di panini per il j-esimo bambino), restituisce TRUE se è possibile preparare i sacchetti, FALSE altrimenti.

Soluzione Si può adottare un approccio greedy secondo cui per preparare i sacchetti si scorrono i sacchetti in input fino a che non si accumulano sufficienti panini per preparare un nuovo sacchetto in output. Se si riescono a preparare tutti i sacchetti in output si restituisce TRUE, altrimenti si restituisce FALSE. Si utilizzano due variabili ausiliarie; `acc` per contare i panini accumulati (ovvero panini in sacchetti in input già considerati ma non ancora inseriti in sacchetti in output), e `j` per indicare il prossimo sacchetto in output da preparare.

```
algoritmo panini(Int IN[k'], Int OUT[k]) --> Bool
    //inizializzazione variabili ausiliarie
    Int acc = 0, j = 1

    //preparazione dei sacchetti
    for i = 1..k' do
        if (acc + IN[i] >= OUT[j]) then
            acc = acc+IN[i]-OUT[j]
            j++
            if (j>k) then return TRUE
        else
            acc = acc+IN[i]
        endif
    endfor
    return FALSE
```

Tutte le operazioni hanno costo costante, quindi la complessità è data dal numero di iterazioni del ciclo. Abbiamo quindi $T(k', k) = O(k')$.

4. Progettare un algoritmo che, dato un grafo non orientato $G = (V, E)$ ed un sottoinsieme $W \subseteq V$, restituisce TRUE se per ogni coppia di vertici $w1, w2 \in W$ esiste un cammino da $w1$ a $w2$, FALSE altrimenti.

Soluzione Tale problema consiste nel verificare che tutti i vertici in W appartengano alla medesima componente connessa. Possiamo quindi selezionare un qualsiasi vertice in $w \in W$, effettuare una visita partendo da w marcando i vertici raggiungibili, e poi semplicemente verificare che tutti i i vertici in W siano stati marcati da tale visita. Si può quindi considerare il seguente algoritmo che utilizza come funzione ausiliaria una visita in profondità.

```
//visita in profondita'
algoritmo DFSvisit(Graph G=(V,E), Node u)
    u.visited = true
    for each v adiacente a u do
        if (not v.mark) then DFS-visit(G,v)
    endfor

algoritmo controllaConnessioni(Graph G=(V,E), Set[Node] W)
    //marcatura di tutti i nodi come non visitati
```

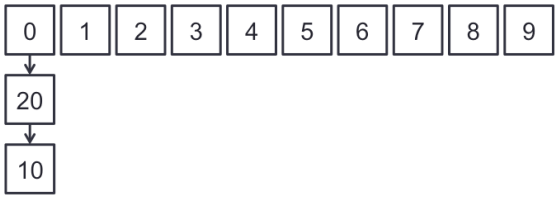
Cont.

```
for each u in V
    u.visited = false
endfor

//DFS a partire da un qualsiasi nodo di W
Node w = W.select()
DFSvisit(G,w)

//verifica che tutti i nodi in W siano stati visitati
for each u in W
    if (not u.visited) then return false
endfor
return true
```

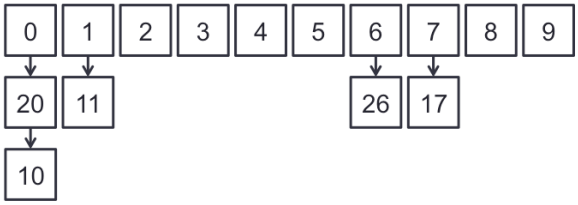
La complessità di tale algoritmo coincide con la complessità dell'algoritmo DFS a cui viene sommato il costo del ciclo for conclusivo (che ha costo lineare, assumendo una implementazione dell'insieme W che permetta di iterare su tutti gli elementi in tempo lineare). Quindi avremo $T(n, m) = O(n + m) + O(n) = O(n + m)$ con n numero di nodi, m numero di archi del grafo in input e assumendo implementazione del grafo tramite liste di adiacenze.

Cont.

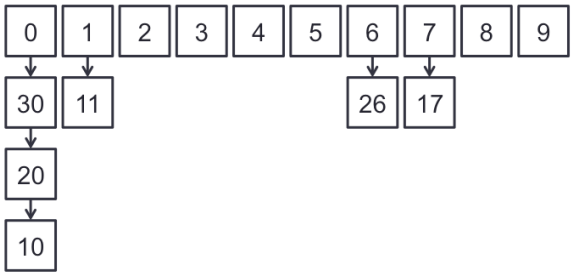
(a) INS(10). La funzione hash ritorna 0 (10 mod 10). Fattore di carico α=1/10



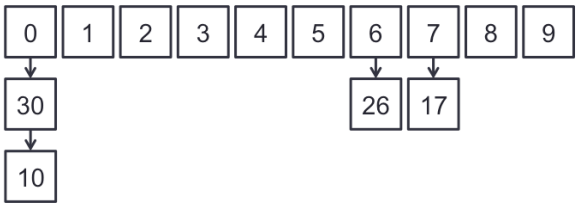
(b) INS(20). La funzione hash ritorna 0. Valore 20 aggiunto in testa alla lista. Fattore di carico α=2/10



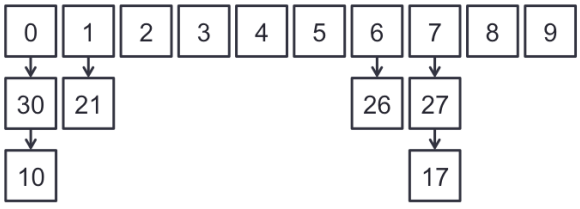
(c) INS(11), INS(17), INS(26). Valori aggiunti in testa alle rispettive liste in posizione 1, 7, 6. Alla fine degli inserimenti α=5/10



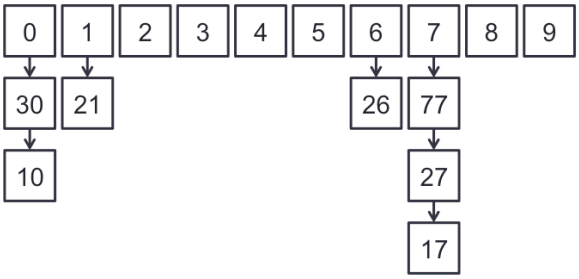
(d) INS(30). La funzione hash ritorna 0. Valore 30 aggiunto in testa alla lista. Fattore di carico α=6/10



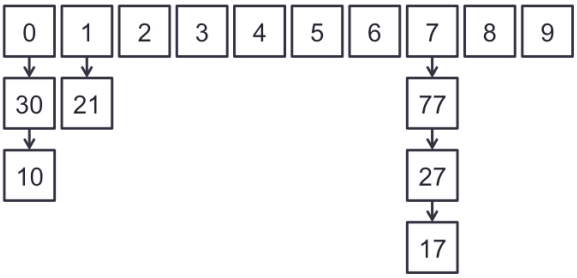
(e) DEL(20), DEL(11). La funzione hash ritorna rispettivamente 0 e 1 e gli elementi sono rimossi dalle liste. Fattore di carico α=4/10



(f) INS(21), INS(27). Valori aggiunti in testa alle rispettive liste in posizione 1 e 7. Alla fine degli inserimenti α=6/10



(g) INS(77). Valore aggiunto in testa alla lista in posizione 7. Fattore di carico α=7/10



(h) DEL(26). Elemento rimosso dalla lista in posizione 6. Fattore di carico α=6/10

The End.